

Multi-priority scheduling algorithm for scientific workflows in cloud

Alaa Albthouse^{1,2}, Farizah Yunus¹, Noor Maizura Mohamad Noor¹

¹Department of Computer Science, Faculty of Ocean Engineering Technology and Informatics, University Malaysia Terengganu, Kuala Terengganu, Terengganu, Malaysia

²Department of Computer Science, Faculty of Information Technology, Al-Hussein Bin Talal University, Ma'an, Jordan

Article Info

Article history:

Received Sep 6, 2023

Revised Feb 2, 2024

Accepted Feb 28, 2024

Keywords:

Cloud computing

Heuristics

Scientific workflows

Task scheduling

Workflow scheduling

ABSTRACT

The public cloud environment has emerged as a promising platform for executing scientific workflows. These executions involve leasing virtual machines (VMs) from public services for the duration of the workflow. The structure of the workflows significantly impacts the performance of any proposed scheduling approach. A task within a workflow cannot begin its execution before receiving all the required data from its preceding tasks. In this paper, we introduce a multi-priority scheduling approach for executing workflow tasks in the cloud. The key component of the proposed approach is a mechanism that logically orders and groups workflow tasks based on their data dependencies and locality. Using the proposed approach, the number of available VMs influences the number of groups (partitions) obtained. Based on the locality of each group's tasks, the priority of each group is determined to reduce the overall execution delay and improve VM utilization. As the results demonstrate, the proposed approach achieves a significant reduction in both execution costs and time in most scenarios.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Alaa Albthouse

Department of Computer Science, University Malaysia Terengganu

Kuala Terengganu, Terengganu, Malaysia

Email: alaa_ahu@yahoo.com

1. INTRODUCTION

The public cloud has emerged as a promising execution environment for resource-intensive applications [1]. An example of such applications is large-scale scientific workflows, which are resource-intensive and cover several application domains, such as physics (LIGO workflows), astronomy (Montage workflow), and bioinformatics (SIPHT workflow) (see Figure 1). Because of the demanding characteristics of these applications, a high-performance computing environment is a crucial requirement for achieving reliable execution duration. In cloud environments, users can rent preconfigured virtual machines (VMs) from public cloud service providers on demand. Scientific workflows consist of hundreds of tasks with data dependency constraints. Using cloud-computing environments, the execution of such workflows can be performed promptly, whereby the VMs required for execution can be leased in an on-demand fashion. Accordingly, to execute a scientific workflow, the user has to determine the number of required VMs and the execution schedule. The number of VMs used and the order of execution influence the efficiency of the obtained schedule. Consequently, increasing the number of VMs increases the execution cost, whereas reducing this number increases the execution time (makespan).

To optimize workflow task execution schedules, several proposals have investigated using nature-inspired optimization techniques such as simulated annealing (SA) and particle swarm optimization (PSO) [2]. Other proposals [3]-[5] have employed heuristic approaches to scheduling workflows. Some of these proposals [6]-[10] have employed greed-based mechanisms to reduce the overall execution time. Others have considered the entire workflow structure, using different strategies to extract structure-related features and establish semi-balanced task groups with similar computational requirements [11]. These proposals are distinguished by the strategies employed to benefit from the extracted structure-related features. Unlike earlier studies, in this work, extracted workflow structure features are used to establish a multi-priority strategy that logically orders the execution schedule of workflow tasks. Using task locality information, the proposed strategy aims to determine an execution schedule that minimizes the execution time and improves the use of VMs.

In this work, we propose a multi-priority scheduling approach to achieve a tradeoff between execution cost and the time of the obtained schedule. The proposed approach divides the workflow into a predetermined number of partitions. The number of partitions can be determined based on the available number of VMs. The partition with the highest priority was designed to include the critical path task, whereby this path is designed to be the longest in terms of the execution time, connecting the exit task to one of the entry-level tasks. This path can be considered the backbone of the workflow, since the execution of its tasks must be prioritized. Figures 1(a) to (d) shows examples of scientific workflows.

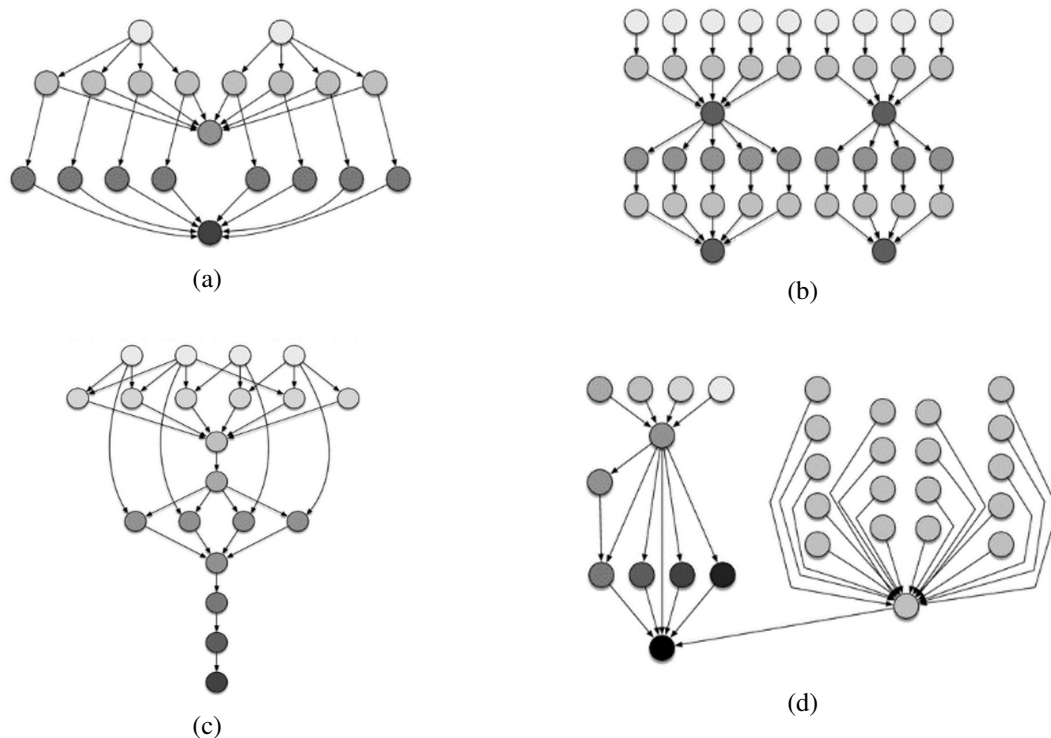


Figure 1. Examples of scientific workflows: (a) cybershake, (b) LIGO, (c) montage, and (d) SIPHT

The remainder of this article is structured as follows. The most closely related literature is discussed in section 2. Section 3 involves details of the system model and problem formulation. The algorithmic solution is presented in section 4, whereas section 5 discusses the experimental settings and results. Finally, the conclusion is presented in section 6.

2. RELATED WORKS

The problem of scheduling scientific workflows in the cloud environment has been extensively studied. Several proposals have addressed the minimization of execution cost and/or time, whereas others have investigated multi-objective optimization criteria that involve balancing the execution load. Accordingly, Ali

et al. [12] introduced a grouping-based algorithm that reduces the total execution time of workflow tasks by categorizing them based on their attributes, such as task dimension and task dependency. The proposed process then determines the execution order of each task's class. In their study, Braun *et al.* [13] used the min-min algorithm for workflow scheduling, prioritizing small tasks and delaying larger tasks for an extended period.

For the same purpose, Kumar and Verma [14] integrated the max-min and min-min algorithms with genetic algorithm functions to enhance the effectiveness of scheduling multiple tasks across multiple virtual machines. Kumar and Verma [14] proposed evolutionary multi-objective optimization (EMO) to simultaneously reduce both the makespan and cost. The authors proposed an encoding scheme that addresses scheduling sequences and task assignments. The authors also proposed a series of new genetic evaluation functions and population operators to tackle this problem. Besides, Manasrah and Ali [15] introduced a hybrid GA-PSO algorithm to decrease the total execution time and cost and balance the workload of interrelated tasks across heterogeneous virtual machines in cloud environments. The suggested algorithm employs the standard deviation to choose the optimal solution, minimizing the distributed load variance over the VMs.

Song *et al.* [16] proposed a new model for scientific workflow scheduling in the cloud. The proposed approach models computationally intensive tasks as assembled tasks and allocates multiple resources to perform them. The authors employed a nested PSO algorithm to enhance the scheduling sequence of tasks and resources. Further, Rodriguez and Buyya [17] developed a PSO algorithm that considered VM boot times to reduce the total workflow tasks execution cost, taking into consideration the deadline constraint in clouds.

Moreover, Kchaou *et al.* [18] proposed a method for task scheduling and data placement to minimize data transfers between cloud data centers. Their proposed approach employs an algorithm rooted in fuzzy clustering, specifically the interval type-2 fuzzy C-means (IT2FCM), along with the meta-heuristic optimization technique known as PSO. The goal is to optimize data movements during the execution of the workflow.

Furthermore, Charrada *et al.* [19] suggested a hybrid approach that determines whether to execute tasks on private or public clouds based on the execution cost, with the assumption that privacy-related constraints are not a concern. For the same purpose, Bossche *et al.* [20] introduced a hybrid approach to minimize the overall execution cost by considering both computational and communication costs and choosing the most cost-efficient cloud that also meets the workflow task's deadline. In their study, Byun *et al.* [21] presented a balanced-time-scheduling algorithm, which calculates the approximate minimum number of virtual machines needed to execute a workflow within a user's deadline while decreasing the execution cost over the entire lifetime of an application. In addition, Wu *et al.* [22] introduced a two-stage approach for DAG applications deployed on the cloud. The proposed method includes a minimal distance algorithm and minimal slack time, aiming to minimize all idle-time slots during the workflow execution while also meeting deadline constraints.

Also, Abrishami *et al.* [23] introduced IaaS cloud partial critical paths (IC-PCPs) to reduce the overall execution cost of workflows while meeting deadline constraints. The authors presented two distinct algorithms for scheduling workflows: a one-phase algorithm that schedules the workflow in a single phase by assigning each partial critical path to a specific instance of a computation service, and a two-phase algorithm that first distributes the overall deadline across the tasks within the workflow and then schedules each task based on its specified sub-deadline. Both algorithms have polynomial time complexity, which makes them suitable for large extensive workflows.

Researchers [3], [5], considered minimizing both the execution time and cost. Almi'ani and Lee [3], suggested a partitioning-based algorithm (PBWS) to determine the priority of each objective using a slack parameter (B); when $B = 0$, the proposed method prioritizes minimizing the total execution time, but when $B = 1$, it focuses on minimizing the overall execution cost. The proposed algorithm follows a sequential partitioning approach, starting with entry-level tasks. The number of VMs allocated per partition is determined based on the B value, ensuring that each task is performed at the earliest starting time (EST) possible. Similarly, the resource demand aware scheduling (RDAS) algorithm proposed in [5] employs a fair allocation strategy to distribute VMs among workflow tasks. The objective is to minimize the makespan and optimize the use of VMs, resulting in reduced overall execution costs.

Sahni and Vidyarthi [24] proposed a just-in-time (JIT-C) method to reduce the total cost of workflow execution under deadline constraints. The proposed algorithm addresses three challenges commonly encountered in cloud platforms: variations in virtual machine performance, delays in acquiring resources, and diverse characteristics of cloud resources. With its potential to address these issues effectively, the algorithm has the potential to become a good choice for integration into cloud resource management.

To reduce the overall execution time, Topcuoglu *et al.* [6] proposed the HEFT algorithm, which works

by initially assigning priority rankings to tasks based on their computational requirements and their workflow locality. Following this prioritization, each task is then scheduled using a VM that minimizes the makespan of the entire workflow. Furthermore, Sandokji and Eassa [25] enhanced the HEFT algorithm by adding a dynamic step that can adjust the scheduling of tasks based on new dynamic scheduling requirements.

As a critical factor in determining the efficiency of the proposed schedule, several proposals have considered the structure of the workflow and the presence of the critical path scheduling solution designs [26], [27]. However, unlike the solutions proposed in the literature, in this work, we use available information to establish a logical order of execution for workflow tasks. An execution schedule is thereby designed to reduce execution time while maximizing the use of VMs.

3. PROBLEM STATMENT

The input to the presented problem in this paper is a workflow represented as a direct acyclic graph (DAG) $G = \langle V, E \rangle$. Set V represents the tasks of the workflows, and set E represents the data interdependence restrictions between the tasks. In addition, we are provided with a set of virtual machines hosted on a public cloud: $VM = vm_1, vm_2, \dots, vm_m$. VMs are divided into two types: R_1 and R_2 . $vm_i \in R_1$ is twice as fast and more expensive than $vm_j \in R_2$. Each task ($v_i \in V$) can only begin its execution once it receives all the necessary data from its parent tasks, and the last parent task to provide data is called the most influential parent (MIP) for v_i . The task has to use either the initial data provided with the workflow or the intermediate data received from its parent tasks to start its execution. When data are transmitted between two tasks, this incurs a communication cost, which is the time required for the data to travel. If two interdependent tasks are assigned to the same virtual machine, the communication cost is considered to be zero.

The objective of the problem presented in this work is to schedule workflow task executions using the available VMs such that the overall execution costs and time are minimized. Decreasing the execution costs means minimizing the total billing cycle of the rented virtual machines, which might increase the execution time. Therefore, the proposed solution in this paper aims to enhance the use of the available VMs to attain the desired objective. By improving the virtual machines' usage, it is expected that the number of billing cycles will decrease, leading to lower costs. Furthermore, improving VM use is also expected to enhance the execution time, as the virtual machines will work more efficiently.

4. ALGORITHMIC SOLUTION

The effectiveness of a scheduling method is influenced by inter-task data dependency constraints. Therefore, the structure of the workflow must be considered when constructing the execution schedule. This section presents a multi-priority scheduling algorithm.

4.1. Multi-priority scheduling algorithm (M-priority)

The multi-priority scheduling algorithm involves three steps: i) attribute extractions, ii) priority calculation, and iii) task scheduling. These three steps operate sequentially to determine the workflow execution schedule.

4.1.1. Attribute extractions

In this step, for each task, the earliest starting time (EST) and the earliest finishing time (EFT) values are calculated (1) and (2). The EST depends on the locality of the task in the workflow, and the EFT depends on the speed of the virtual machine used (R_1 or R_2). For a given task ($v_i \in V$), the EST and the EFT are calculated as (1) and (2):

$$EST(v_i) = \begin{cases} 0 & v_i \text{ is entry task} \\ EFT(MIP_i) & \text{otherwise} \end{cases} \quad (1)$$

$$EFT(v_i) = EST(v_i) + r(v_i) \quad (2)$$

where MIP_i refers to the most influential parent for task v_i , and $r(v_i)$ denotes the running time of task v_i . Once the EST and EFT for each task are calculated, these values are used to calculate the latest starting time (LST) and the latest finishing time (LFT) (3) and (4). These two values are introduced to determine if the

execution time of any given task can be delayed without impacting the overall execution time. The calculation of these values starts with the exit task, and values are obtained as (3) and (4):

$$LST(v_i) = \begin{cases} EST(v_i) & v_i \text{ is exit task} \\ MIN(EST(ANS_i)) & otherwise \end{cases} \quad (3)$$

$$LFT(v_i) = LST(v_i) + r(v_i) \quad (4)$$

where $MIN(EST(ANS_i))$ denotes the minimum EST for task v_i 's direct ancestor; a direct ancestor for task v_i is a task that is connected to v_i via a single edge. As discussed, the LST and LFT represent a safe gap that can be used to determine if a given task must be scheduled without any delay.

4.1.2. Priority calculation

This step starts with calculating the allowable execution delay for each task ($d(v_i)$); this value is calculated as (5):

$$d(v_i) = LST(v_i) - EST(v_i) \quad (5)$$

The $d \geq 0$ value of each task impacts its execution order. For instance, in situations where a task has a zero d value, the execution of this task has to be prioritized. Accordingly, this step uses the delay value to calculate the task priority values. From a priority perspective, the workflow tasks are divided into a set of partitions based on their priorities, where the partition ID number represents its task priority. The number of partitions used is a parameter that depends on the available number of VMs. The first partition (pr_1) is designed to host the tasks with the highest priorities ($d=0$). Regarding each task with a delay of $d > 0$, the identity of its partition is calculated by first determining the partition delay range as (6):

$$n_d = \left\lceil \frac{MAX(d)}{(NP - 1)} \right\rceil \quad (6)$$

where $MAX(d)$ refers to the maximum allowable delay for a task, and NP denotes the number of partitions. By dividing the maximum delay by the number of partitions, n_d represents the time window for each partition in terms of delay. For instance, if the value of $MAX(d)$ is 20, and the number of partitions is 3, the value of n_d is 10. This range value denotes that, in any partition, the difference between any two tasks in terms of delay should be < 10 . Once the value of n_d is calculated, a task ($v_i \in V$) partition ID is determined as (7):

$$pr(v_i) = \left\lfloor \frac{d(v_i)}{(n_d)} \right\rfloor \quad (7)$$

accordingly, a task priority value is controlled by its delay value, d , and the number of partitions used, where increasing the value of d results in reducing the task priority.

4.1.3. Task scheduling

Algorithm 1 illustrates the task-scheduling step process. Scheduling workflow tasks is performed level-by-level in a sequential fashion, where the entry-level tasks are the first to be scheduled. Each level's tasks are ordered based on their priority value. In situations where multiple tasks share the same priority value, these tasks will be ordered based on their EST. Scheduling tasks is performed using a greedy strategy, where a task is scheduled on the first VM that results in an actual starting time less than or equal to the LST. During the process of scheduling tasks, the VMs with the lowest speeds (R_1) are the first to be examined. Accordingly, if no VM from the standard category can satisfy the starting time constraints, the VMs with the highest speed (R_2) will also be considered for task executions. Delaying the use of high-speed VMs reduces the execution cost, since the cost of using this type of VM is typically higher.

4.2. Discussion and limitations

To clarify the process of the presented approach, consider the example shown in Figures 2(a) and (b). The task characteristics are shown in Table 1. In this example, for simplicity, we assume that we have a single VM from R_1 and a single VM from R_2 . Additionally, the number of partitions used is limited to two. Once the priority calculation is performed, the tasks with a priority equal to one are identified to be 1,4 and 6,8, whereby the reset values of the tasks are assigned a priority equal to two.

Algorithm 1 Task scheduling**Input:** G, R (available VMs)**Output:** S (schedule)

```

1: procedure TASKSCHEDULING( $G, R$ )
2:    $L \leftarrow \text{sortTasksBasedonLevel}(G)$ 
3:   for  $l_i \in L$  do
4:      $V' \leftarrow \text{orderBasedonPriorityandEST}(l_i)$ 
5:     for  $v_i \in V'$  do
6:        $r \leftarrow \text{findVM}(v_i, R_1)$ 
7:       if  $r$  is empty then
8:          $r \leftarrow \text{findVM}(v_i, R_2)$ 
9:       add  $\langle r, v_i \rangle$  to  $S$ 
10:  return  $S$ 

```

At the beginning of the task-scheduling step, the entry-level tasks are considered for scheduling. Task v_1 is scheduled on $vm_1 \in R_1$, and task v_2 is scheduled on $vm_2 \in R_2$. Then, while scheduling the other levels' tasks, vm_1 is scheduled to execute tasks v_4, v_6 , and v_8 , whereas tasks v_3, v_5 , and v_7 are scheduled to be executed on vm_2 . In this example, the execution time (makespan) is 66s. Based on this example, we can see the importance of considering VMs with the lowest speeds as the first option for scheduling tasks, since all task attribute values are calculated based on this speed. Additionally, speeding up the execution of the tasks with the highest priority will not help reduce the makespan since these tasks depend on data generated by lower-priority tasks. The ability to bind the workflow makespan with the LFT of the exit task is impacted by the number and the speed of the available VMs, in situations where such bounding is mandatory, the following condition must be satisfied for each workflow-level task:

$$\frac{r(l_i) - r(pr_1)}{|R_1| - 1 + |R_2| \times 2} \leq r(pr_1) \quad (8)$$

where $r(pr_1)$ denotes the running time of the task with priority one located at level l_i , and $r(l_i)$ refers to the running time of the entire level's tasks. Additionally, $|R_1|$ refers to the number of available vms from the R_1 category, and $|R_2|$ refers to the number of available vms from the R_2 type. In this calculation, the number of vms from type R_1 is reduced by one, since a single VM of this type is reserved to execute the task with a priority equal to one. In situations where workflow tasks are divided into more than two partitions, the same condition will hold since priority-one tasks influence the rest of the task executions.

The available number of VMs impacts the efficiency of the proposed approach. To clarify this dependency, consider the example shown in Figure 2. In this example, if the number of available VMs is increased to three, the M-Priority will obtain a better schedule since v_3 will be assigned to a third group, which allows the execution of task v_3 any delay.



Figure 2. Workflow scheduling example: (a) input workflow and (b) priority calculation

Table 1. Attribute calculation

Task Id	Runtime	MIP	EST	EFT	LST	LFT	d
1	15	0	0	15	0	15	0
2	12	0	0	12	3	15	3
3	16	1	15	31	21	37	7
4	22	1	15	37	15	37	0
5	14	2	12	26	23	37	11
6	18	4	37	55	37	55	0
7	13	4	37	50	42	55	5
8	11	6	55	66	55	66	0

Regarding the communication cost, although the proposed approach does not consider this cost during the scheduled construction, considering this cost is not expected to reduce the overall performance of the presented approach. The M-Priority aims to reduce the data dependency between the constructed groups. Therefore, data-dependency tasks will likely to be executed on the same VM, and communication costs between tasks assigned to the same VM is zero. Furthermore, when the workflow has a balanced structure, the efficiency of the proposed approach is expected to increase since the computational and communication requirements of the obtained groups are expected to be the same.

In dynamic environments, the presented approach is expected to be highly impacted by the probability of hardware failure. In the M-Priority, VMs are divided between the groups based on their computational requirements. Thus, when hardware failure occurs, the performance of the proposed approach might be degraded since groups that lost some of their assigned VMs might create a performance bottleneck.

5. RESULTS AND DISCUSSION

This section provides a detailed discussion of the experimental settings and outcomes used to evaluate the effectiveness of the proposed multi-priority scheduling algorithm (M-priority). In the presented results, four real-world applications (LIGO, CyberShake, Montage, and SIPHT [28]) were used as input workflows. Each of the workflows used consisted of 1000 tasks, and each experiment was repeated 10 times using 10 different workflows (the average is displayed). The experiments used two types of VMs: R_1 and R_2 . R_2 was designed to be twice as fast as R_1 , and the charge rates of R_1 and R_2 were 0.10 and 0.20 USD per hour, respectively. In line with [1], to evaluate the performance of the presented algorithm, we used the following evaluation metrics:

- Makespan (m): the makespan represents the total time taken to complete all workflow tasks, and it is indicated by the AFT (actual finishing time) of the final task being executed (also known as the exit task (v_e)). The makespan can be expressed as $m = AFT(v_e)$;
- Cost (c): cost refers to the total expenses incurred for using the VMs, and it is determined using the billing cycle formulation (bc), which is based on an hourly rate. The cost (c) is calculated as (9):

$$c = \sum_{vm_i \in R_1} bc(vm_i) \times 10 + \sum_{vm_j \in R_2} bc(vm_j) \times 20 \quad (9)$$

To validate the effectiveness of the proposed algorithm, comparisons were made against four benchmark algorithms from the literature namely, HBA [1], RDAS [8], PBWS [3], and HEFT [6]. The HBA algorithm applies a lookahead strategy that aims to minimize the data dependency between the constructed partitions. The RDAS algorithm partitions workflow tasks based on the workflow's structure and divides the available VMs between the partitions using a fair division strategy to minimize both the cost and makespan. The PBWS algorithm determines the number of VMs and the execution schedule based on predetermined weights for each objective (makespan or cost). The HEFT algorithm employs a greedy approach to sequentially schedule tasks, selecting the VM that can minimize the AFT of each task. To ensure a fair comparison, the PBWS algorithm was run first, and the number of VMs used by the PBWS algorithm was then used as input for the M-priority, HBA, RDAS, and HEFT algorithms. To determine the number of partitions used by the M-priority algorithm, we performed a sensitivity analysis. Based on the number of available VMs, two partitions achieved the best performance. Accordingly, in the presented experiments, two partitions were used by the M-priority algorithm. The results of the performed experiments are summarized in Table 2.

Table 2. Results summary

Workflow	Algorithm	Makespan	Cost
LIGO	M-priority	1495	1640
	PBWS	1594	4659
	RDAS	1643	4243
	HEFT	1300	4974
	HBA	1578	1640
CyberShake	M-priority	692	380
	PBWS	550	294
	RDAS	750	541
	HEFT	712	516
	HBA	533	315
Montage	M-priority	211	1500
	PBWS	956	497
	RDAS	720	410
	HEFT	815	446
	HBA	242	1500
SIPHT	M-priority	3163	1760
	PBWS	8465	3282
	RDAS	6991	2862
	HEFT	7516	2974
	HBA	4464	2320

LIGO: Figures 3(a) and (b) shows that the M-priority algorithm demonstrated a significant improvement in terms of cost compared with other algorithms. This is due to the priority mechanism employed by the M-priority algorithm, which works by scheduling tasks based on their data dependency constraints. Such mechanisms improve the use of VMs. In addition, the M-priority algorithm outperforms the PBWS, HBA and RDAS algorithms in terms of makespan, wherein the best performance is achieved by the HEFT algorithm. To understand the factors behind this performance, let us consider the structure of the LIGO workflow. The structure of this workflow can be considered balanced, with the middle level introducing a high level of dependency. The presence of this task level limits the benefit of the M-priority algorithm in terms of the makespan since it reduces the overall delay value, $\sum_{v_i \in V} d(v_i)$.

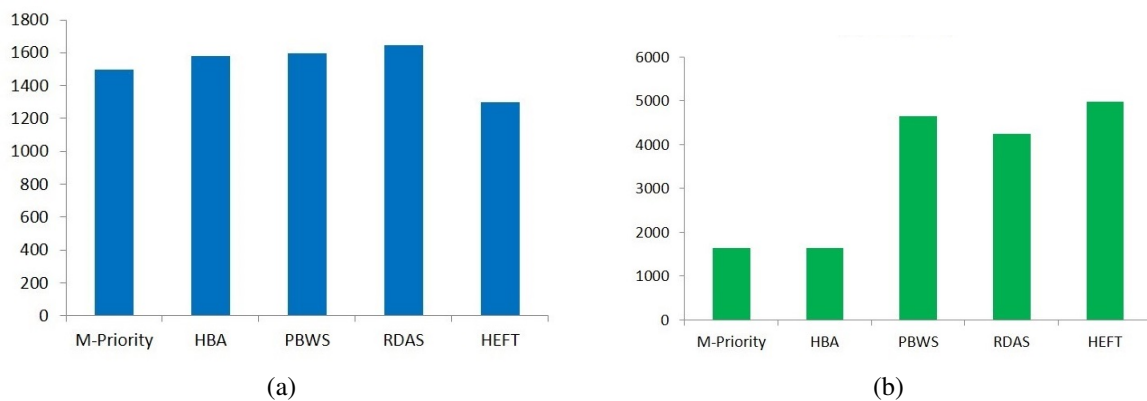


Figure 3. LIGO workflow experiment results: (a) makespan and (b) cost

CyberShake: Figures 4(a) and (b) shows the results achieved by the M-priority algorithm compared with the other benchmark algorithms. These results show that the proposed algorithm achieved a 20% cost reduction and a 5% reduction in execution time compared with the RDAS and HEFT algorithms. However, the PBWS and the HBA algorithms managed to achieve a better performance compared with the M-priority algorithm. The CyberShake workflow had a shallow structure, which resulted in it having a significant number of tasks located at some of its levels. Such a structure increases the delay sensitivity of the M-priority algorithm, since tasks with small execution time windows are expected to degrade the M-priority algorithm.

Montage: the internal structure of the Montage workflow is not balanced; the number of tasks located in the workflow levels varies significantly. This structure increased the execution cost of the schedule obtained by the M-priority algorithm, since not all of the VMs were fully utilized. In contrast, in terms of the makespan,

such a structure highlights the benefits of the delay strategy employed by the M-priority algorithm, since it established an efficient logical order of execution for the workflow tasks. Such factors are responsible for the performance shown in Figures 5(a) and (b), whereby prioritizing the execution of tasks with high priority helped reduce the overall execution time.

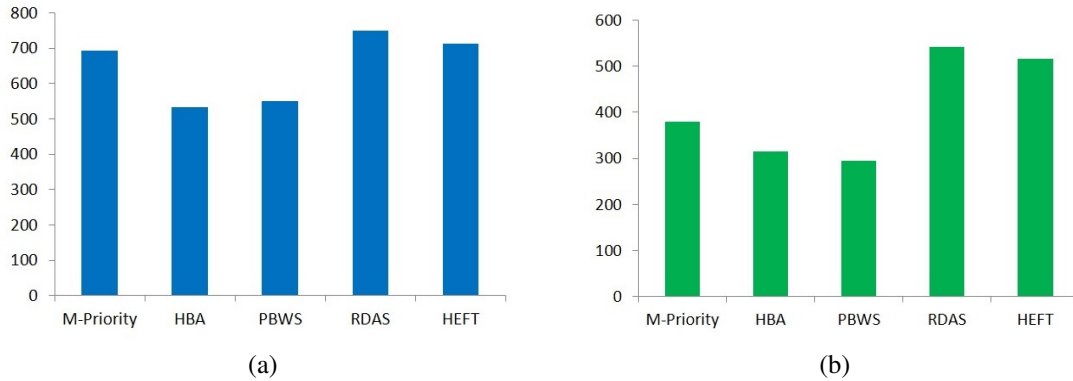


Figure 4. Cybershake workflow experiment results: (a) makespan and (b) cost

SIPHT: as presented in Figures 6(a) and (b), the M-priority algorithm significantly outperformed all other algorithms in the SHIPT workflow experiment. The results show that the M-priority algorithm achieved a time and cost reduction of over 20% compared with other algorithms. The SIPHT workflow also had an unbalanced structure, which resulted in emphasizing the advantages of the mechanism employed by the M-priority algorithm in reducing the execution time. Regarding the execution cost, in the SIPHT workflow, the lower level had the highest number of tasks, and this increased the use of VMs.

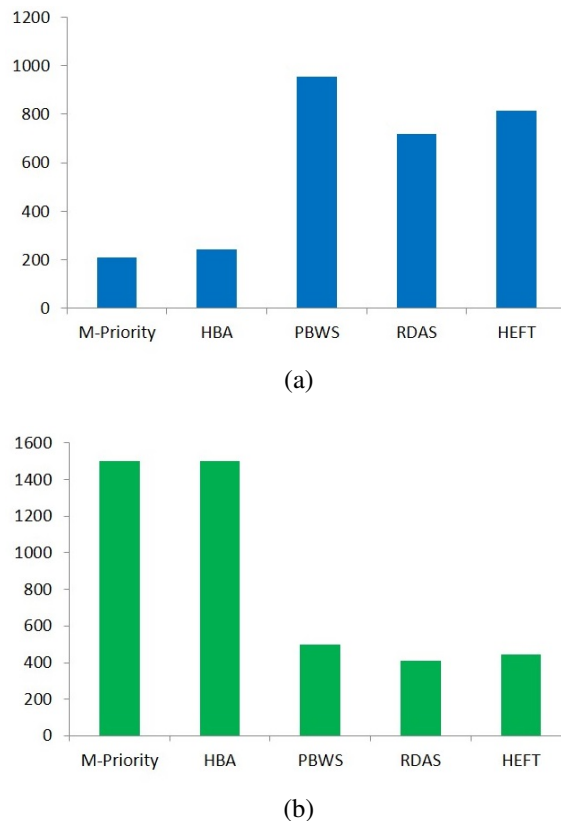


Figure 5. Montage workflow experiment results: (a) makespan and (b) cost

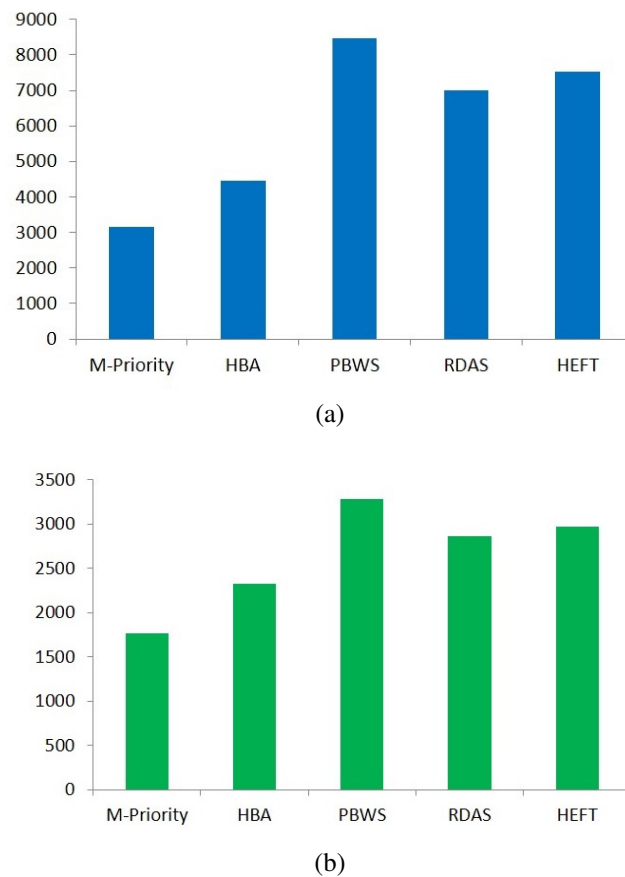


Figure 6. SIPHT workflow experiment results: (a) makespan and (b) cost

6. CONCLUSION

Data dependency in workflow tasks plays a critical role in determining the efficiency of a resulting schedule. In this work, we presented a multi-priority scheduling approach that establishes a logical order of execution for workflow tasks. The method groups workflow tasks based on their locality, with the priority of each partition determined based on its task's order of execution. The results show that the number of available VMs influences the schedule's performance, as having a relatively high number of VMs highlights the advantages of the approach. In future work, we plan to incorporate communication costs into the proposed approach to determine its performance in more realistic settings. Additionally, we plan to expand the proposed method to address dynamic scheduling scenarios, where the availability of resources and the priority of tasks may change dynamically. We also plan to enhance our method to better handle such dynamic changes.




REFERENCES

- [1] A. Albtoush, F. Yunus, K. Almi'ani, and N. M. Noor, "Structure-aware scheduling methods for scientific workflows in cloud," *Applied Sciences*, vol. 13, no. 3, p. 1980, Feb. 2023, doi:10.3390/app13031980.
- [2] M. Farid, R. Latip, M. Hussin, and N. A. W. Abdul Hamid, "Scheduling Scientific Workflow Using Multi-Objective Algorithm With Fuzzy Resource Utilization in Multi-Cloud Environment," in *IEEE Access*, vol. 8, pp. 24309-24322, 2020, doi: 10.1109/ACCESS.2020.2970475.
- [3] K. Almi'ani and Y. C. Lee, "Partitioning-Based Workflow Scheduling in Clouds," *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, 2016, pp. 645-652, doi: 10.1109/AINA.2016.83.
- [4] A. Pasdar, Y. C. Lee, and K. Almi'ani, "Toward cost efficient cloud bursting," *Service-Oriented Computing*, pp. 299-313, 2019, doi:10.1007/978-3-030-33702-5_23.
- [5] K. Almi'ani, Y. C. Lee, and B. Mans, "Resource demand aware scheduling for workflows in clouds," *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, 2017, pp. 1-5, doi: 10.1109/NCA.2017.8171368.




- [6] H. Topcuoglu, S. Hariri, and M. -Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, March 2002, doi: 10.1109/71.993206.
- [7] R. N. Calheiros and R. Buyya, "Cost-effective provisioning and scheduling of deadline-constrained applications in hybrid clouds," *Web Information Systems Engineering - WISE 2012*, pp. 171-184, 2012, doi:10.1007/978-3-642-35063-4_13.
- [8] K. Almi'ani, Y. C. Lee, and B. Mans, "On efficient resource use for scientific workflows in clouds," *Computer Networks*, vol. 146, pp. 232-242, Dec. 2018, doi:10.1016/j.comnet.2018.10.003.
- [9] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158-169, Jan. 2013, doi:10.1016/j.future.2012.05.004.
- [10] A. Albthouse, N. M. M. Noor and F. Yunus, "Utility-based Scheduling Solution for Scientific Workflow on Cloud," *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, 2021, pp. 1-6, doi: 10.1109/ISNCC52172.2021.9615698.
- [11] G. K. Toussi, M. Naghibzadeh, S. Abrishami, H. Taheri, and H. Abrishami, "EDQWS: An enhanced divide and conquer algorithm for workflow scheduling in cloud," *Journal of Cloud Computing*, vol. 11, no. 1, May 2022, doi:10.1186/s13677-022-00284-8.
- [12] H. G. E. D. H. Ali, I. A. Saroit, and A. M. Kotb, "Grouped tasks scheduling algorithm based on QoS in cloud computing network," *Egyptian Informatics Journal*, vol. 18, no. 1, pp. 11-19, Mar. 2017, doi:10.1016/j.eij.2016.07.002.
- [13] T. D. Braun et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810-837, Jun. 2001, doi:10.1006/jpdc.2000.1714.
- [14] P. Kumar and A. Verma, "Scheduling using improved genetic algorithm in cloud computing for independent tasks," *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, Aug. 2012, doi:10.1145/2345396.2345420.
- [15] A. M. Manasrah and H. B. Ali, "Workflow scheduling using hybrid GA-PSO algorithm in cloud computing," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1-16, 2018, doi:10.1155/2018/1934784.
- [16] A. Song, W. -N. Chen, X. Luo, Z. -H. Zhan, and J. Zhang, "Scheduling Workflows With Composite Tasks: A Nested Particle Swarm Optimization Approach," in *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 1074-1088, 1 March-April 2022, doi: 10.1109/TSC.2020.2975774.
- [17] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," in *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222-235, 1 April-June 2014, doi: 10.1109/TCC.2014.2314655.
- [18] H. Kchaou, Z. Kechaou, and A. M. Alimi, "A PSO task scheduling and IT2FCM Fuzzy data placement strategy for scientific cloud workflows," *Journal of Computational Science*, vol. 64, p. 101840, Oct. 2022, doi:10.1016/j.jocs.2022.101840.
- [19] F. B. Charrada and S. Tata, "An Efficient Algorithm for the Bursting of Service-Based Applications in Hybrid Clouds," in *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 357-367, 1 May-June 2016, doi: 10.1109/TSC.2015.2396076.
- [20] R. V. Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds," *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, 2011, pp. 320-327, doi: 10.1109/CloudCom.2011.50.
- [21] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of Elastic Resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011-1026, Oct. 2011, doi:10.1016/j.future.2011.05.001.
- [22] H. Wu, X. Hua, Z. Li and S. Ren, "Resource and Instance Hour Minimization for Deadline Constrained DAG Applications Using Computer Clouds," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 885-899, 1 March 2016, doi: 10.1109/TPDS.2015.2411257.
- [23] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158-169, Jan. 2013, doi:10.1016/j.future.2012.05.004.
- [24] J. Sahni and D. P. Vidyarthi, "A Cost-Effective Deadline-Constrained Dynamic Scheduling Algorithm for Scientific Workflows in a Cloud Environment," in *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 2-18, 1 Jan.-March 2018, doi: 10.1109/TCC.2015.2451649.
- [25] S. Sandokji and F. Eassa, "Dynamic variant rank heft task scheduling algorithm toward Exascale Computing," *Procedia Computer Science*, vol. 163, pp. 482-493, 2019, doi:10.1016/j.procs.2019.12.131.
- [26] L. Teylo, U. de Paula, Y. Frota, D. de Oliveira, and L. M. A. Drummond, "A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds," *Future Generation Computer Systems*, vol. 76, pp. 1-17, Nov. 2017, doi:10.1016/j.future.2017.05.017.
- [27] M. Farid, R. Latip, M. Hussin, and N. A. W. Abdul Hamid, "Scheduling Scientific Workflow Using Multi-Objective Algorithm With Fuzzy Resource Utilization in Multi-Cloud Environment," in *IEEE Access*, vol. 8, pp. 24309-24322, 2020, doi: 10.1109/ACCESS.2020.2970475.
- [28] S. Kaur, P. Bagga, R. Hans, and H. Kaur, "Quality of service (qos) aware workflow scheduling (WFS) in Cloud computing: A systematic review," *Arabian Journal for Science and Engineering*, vol. 44, no. 4, pp. 2867-2897, Nov. 2018. doi:10.1007/s13369-018-3614-3.

BIOGRAPHIES OF AUTHORS






Alaa Albthouseh    is currently a lecturer in the Information Technology School of Computer Science at the University of Al-Hussein Bin Talal, Ma'an, Jordan where he has been a faculty member since 2010. He received the B.Sc. degree in computer science from Mu'tah University, Al-Karak, Jordan, in 2001, and the M.Sc. degree in computer science from AABFS University, Amman, Jordan, in 2009, is currently a Ph.D. student in the Department of Computer Sciences, University Malaysia Terengganu, Kuala Terengganu, Malaysia. His research interests include the cloud computing and workflow scheduling. He can be contacted at email: alaa_ahu@yahoo.com.



Farizah Binti Yunus    received her B.Sc. degree in electrical engineering (telecommunication) and Ph.D. degrees in telecommunication engineering from Universiti Teknologi Malaysia (UTM). She is a Senior Lecturer of Computer Science at Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu (UMT). Her research interests include wireless sensor network, networking, cloud computing, and internet of things (IoT). She is a member of MBOT and BEM. She has worked as researcher in several national funded R&D projects. She can be contacted at email: farizah.yunus@umt.edu.my.



Noor Maizura Mohamad Noor    obtained her Diploma and Bachelor of Computer Science from Universiti Pertanian Malaysia, Serdang Selangor in 1991 and 1994 respectively. She earned her Master of Science (Computer Science) from Universiti Putra Malaysia in 1997. Later in 2005, she acquired her doctoral degree in Computer Science from the University of Manchester, the United Kingdom. Her outstanding accomplishments led to her appointment as a professor in 2017. Her recent research work focuses on improving organizational decision-making practices using technologies. This includes research interests in the design, development, and evaluation of decision support systems for analyzing and improving decision processes. Her research interests also focus on the areas of computer science, intelligent decision support systems, clinical decision support systems, and information systems. She has presented and published over two hundred of papers on the decision support system at various international and local refereed journals, conferences, seminars, and symposiums. She can be contacted at email: maizura@umt.edu.my.